МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Алгоритмы и структуры данных

Тема: Бинарные деревья

Вариант 13-д

Студент гр. 8304 _____ Барышев А.А.

Преподаватель _____ Фирсов К.В.

Санкт-Петербург

2019

Задание.

< формула > ::= < терминал > | (< формула > < знак > < формула >)

<знак>::=+|-|*

<терминал>::=0|1|...|9|a|b|...|z

можно представить в виде бинарного дерева («дерева-формулы») с элементами типа Elem=char согласно следующим правилам:

- формула из одного терминала представляется деревом из одной вершины с этим терминалом;
- формула вида (f1 s f2) представляется деревом, в котором корень~ это знак s, а левое и правое поддеревья~ соответствующие представления формул f1 и f2.

Требуется:

- преобразовать дерево-формулу t, заменяя в нем все поддеревья, соответствующие формулам ((f1 * f2) + (f1 * f3)) и ((f1 * f3) + (f2 * f3)), на поддеревья, соответствующие формулам (f1 * (f2 + f3)) и ((f1 + f2) * f3);
- с помощью построения дерева-формулы t преобразовать заданную формулу f из постфиксной формы (перечисление узлов в порядке ЛПК) в инфиксную.

Цель работы.

Научиться обрабатывать АТД бинарное дерево с помощью его стандартных методов, так как root, right, left. Применить полученные знания на практике, решив задание.

Описание програмы.

Данная программа принимает на вход текст из текстового файла, который заканчивается строкой END. Программа принимает строку, в которой ищет «знаки», а находя их, проверяет, есть ли рядом со знаком скобки. Если есть, то происходит рекурсивный вызов данной функции, а если нет, то сразу создаётся бинарное дерево, на которое ставится указатель.

Обход дерева для его преобразования осуществляется по принципу КЛП.

Функции.

Основные функции данной программы следующие:

binTree EnterBinTreeFormula(std::string, int);

void replaceElemBinTree(binTree);

Функция EnterBinTreeFormula позволяет вводить бинарное дерево в виде указанной в задании формулы. Она рекурсивно передаёт позицию открывающихся скобок, чтобы внутри них находить «знак», поскольку возле знака могут быть только скобки или терминал.

Функция replaceElemBinTree позволяет обходить бинарное дерево по принципу КЛП, находя в нём участки, которые возможно преобразовать. Все преобразования осуществляются с помощью стандартных методов АТД бинарного дерева.

Тесты.

Данная программа тестировалась на всевозможных верных наборах данных(соответствующих условию). Если входная строка условию не соответствует, то программа будет уходить в бесконечный цикл или выдавать Seg fault. Тесты приведены ниже:

Преобразование дерева для формулы: ((f*c)+(c*s))

Бинарное дерево (повернутое):

```
 + * s
     c
   * c
     f
```

Бинарное дерево (повернутое):

```
 * c
   + s
     f
```

Постфиксная форма: fs+c*

****

Преобразование дерева для формулы: (((b-c)*c)+(c*s))

Бинарное дерево (повернутое):

```
 + * s
     c
   * c
     - c
       b
```

Бинарное дерево (повернутое):

```
 * c
   + s
     - c
       b
```

Постфиксная форма: bc-s+c*

****

Преобразование дерева для формулы: (7+(c*s))

Бинарное дерево (повернутое):

```
 + * s
     c
   7
```

Бинарное дерево (повернутое):

```
 + * s
     c
   7
```

Постфиксная форма: 7cs*+

Преобразование дерева для формулы: (((d*c)-3)*(x+(g-(b*d))))

Бинарное дерево (повернутое):

```
 * + - * d
       b
     g
   x
  - 3
    * c
      d
```

Бинарное дерево (повернутое):

```
 * + - * d
       b
     g
   x
  - 3
    * c
      d
```

Постфиксная форма: dc*3-xgbd*-+*

****

Выводы.

Оценивается степень соответствия полученных результатов расчетов и экспериментов с теоретическими данными.

Дается объяснение полученных в ходе работы зависимостей и результатов.

КОД ПРОГРАММЫ.

```
#INCLUDE <IOSTREAM>
#INCLUDE <FSTREAM>
#INCLUDE <FSTREAM>
#INCLUDE <CSTDLIB>

STD::IFSTREAM INFILE("TESTS/TESTS.TXT");
STD::OFSTREAM FOUT;
TYPEDEF CHAR BASE;

STRUCT NODE {
     BASE INFO;
     NODE *LT;
     NODE *RT;
     NODE() {
          LT = NULL;
          RT = NULL;
     }
};

TYPEDEF NODE *BINTREE;

BOOL ISNULL(BINTREE);
BASE ROOTBT(BINTREE);
BINTREE CREATE(VOID);
BINTREE LEFT(BINTREE);
BINTREE RIGHT(BINTREE);
BINTREE CONSBT(CONST BASE &X, BINTREE &LST,  BINTREE &RST);
UNSIGNED INT SIZEBT(BINTREE B);
VOID DESTROY(BINTREE&);
VOID OUTBT(BINTREE B);
VOID DISPLAYBT(BINTREE B, INT N);
VOID PRINTKLP(BINTREE B);
VOID PRINTLKP(BINTREE B);
```

```
    VOID PRINTLPK(BINTREE B);

    BINTREE ENTERBINTREEFORMULA(STD::STRING STRBINTREE, INT
POSITION){
        BINTREE LEFT, RIGHT;
        INT BUF = 0;
        INT COUNTER = 0;
        WHILE( !((STRBINTREE[POSITION] == '*' ||
STRBINTREE[POSITION] == '-' || STRBINTREE[POSITION] == '+') &&
COUNTER == 0) ){
            POSITION++;
            IF(STRBINTREE[POSITION] == '(')
                COUNTER++;
            ELSE IF((STRBINTREE[POSITION] == '*' ||
STRBINTREE[POSITION] == '+' || STRBINTREE[POSITION] == '-') &&
COUNTER > 0){
                COUNTER--;
                POSITION++;
                IF(STRBINTREE[POSITION] == '('){
                    COUNTER++;
                }
            }
        }
        BUF = POSITION;
        INT BUF1 = BUF;
        COUNTER = 0;
        IF(STRBINTREE[BUF1 - 1] == ')'){
            BUF1--;
            WHILE( !((STRBINTREE[BUF1] == '(' && COUNTER
== 0) ){
                BUF1--;
                IF(STRBINTREE[BUF1] == ')'){
                    COUNTER++;
                }
```

```
                    ELSE IF((STRBINTREE[BUF1] == '(' && COUNTER
> 0){
                         COUNTER--;
                         BUF1--;
                         IF(STRBINTREE[BUF1] == ')'){
                              COUNTER++;
                         }
                    }
               }
               LEFT = ENTERBINTREEFORMULA(STRBINTREE, BUF1);
          }
          ELSE {
               BINTREE P1;
               P1 = NEW NODE;
               P1->INFO = STRBINTREE[BUF - 1];
               P1->LT = NULL;
               P1->RT = NULL;
               LEFT = P1;
          }
          IF(STRBINTREE[BUF + 1] == '('){
               RIGHT = ENTERBINTREEFORMULA(STRBINTREE, BUF + 1);
          }
          ELSE {
               BINTREE P2;
               P2 = NEW NODE;
               P2->INFO = STRBINTREE[BUF + 1];
               P2->LT = NULL;
               P2->RT = NULL;
               RIGHT = P2;
          }
          RETURN CONSBT(STRBINTREE[POSITION], LEFT, RIGHT);
     }

     BINTREE CONSBT(CONST BASE &X, BINTREE &LST, BINTREE &RST) {
          BINTREE P;
```

```
     P = NEW NODE;
     IF (P != NULL) {
          P->INFO = X;
          P->LT = LST;
          P->RT = RST;
          RETURN P;
     }
     ELSE {
          STD::CERR << "MEMORY NOT ENOUGH\N";
          EXIT(1);
     }
}

VOID OUTBT(BINTREE B) {
     IF (B!=NULL) {
          FOUT << ROOTBT(B);
          OUTBT(LEFT(B));
          OUTBT(RIGHT(B));
     }
     ELSE
          FOUT << '/';
}

VOID DISPLAYBT (BINTREE B, INT N) {
     IF (B!=NULL) {
          FOUT << ' ' << ROOTBT(B);
          IF(!ISNULL(RIGHT(B))) {
               DISPLAYBT (RIGHT(B),N+1);
          }
          ELSE
               FOUT << STD::ENDL;
          IF(!ISNULL(LEFT(B))) {
               FOR(INT I = 1; I <= N; I++)
                    FOUT << "   ";
               DISPLAYBT (LEFT(B),N+1);
```

```
          }
     }
}

UNSIGNED INT SIZEBT (BINTREE B)
{
     IF (ISNULL(B)) RETURN 0;
     ELSE RETURN SIZEBT (LEFT(B))+ SIZEBT(RIGHT(B)) + 1;
}

VOID PRINTKLP (BINTREE B)
{    IF (!ISNULL(B)) {
          FOUT << ROOTBT(B);
          PRINTKLP (LEFT(B));
          PRINTKLP (RIGHT(B));
     }
}

VOID PRINTLKP (BINTREE B)
{    IF (!ISNULL(B)) {
          PRINTLKP (LEFT(B));
          FOUT << ROOTBT(B);
          PRINTLKP (RIGHT(B));
     }
}

VOID PRINTLPK (BINTREE B)
{    IF (!ISNULL(B)) {
          PRINTLPK (LEFT(B));
          PRINTLPK (RIGHT(B));
          FOUT << ROOTBT(B);
     }
}
```

```
BINTREE CREATE() {
     RETURN NULL;
}

BOOL ISNULL(BINTREE B) {
     RETURN (B == NULL);
}

BASE ROOTBT (BINTREE B)
{    IF (B == NULL) { STD::CERR << "ERROR: ROOTBT(NULL) \N";
EXIT(1); }
     ELSE RETURN B->INFO;
}

BINTREE LEFT (BINTREE B)
{    IF (B == NULL) { STD::CERR << "ERROR: LEFT(NULL) \N";
EXIT(1); }
     ELSE RETURN B ->LT;
}

BINTREE RIGHT (BINTREE B)
{    IF (B == NULL) { STD::CERR << "ERROR: RIGHT(NULL) \N";
EXIT(1); }
     ELSE RETURN B->RT;
}

VOID DESTROY (BINTREE &B) {
     IF (B != NULL){
          DESTROY (B->LT);
          DESTROY (B->RT);
          DELETE B;
          B = NULL;
     }
}
```

```
     VOID REPLACEELEMBINTREE(BINTREE B)
     {    IF (!ISNULL(B)) {
               IF(ROOTBT(B) == '+' || ROOTBT(B) == '-'){
                    IF(ROOTBT(LEFT(B)) == '*' && ROOTBT(RIGHT(B))
== '*'){
                         IF(ROOTBT(B->LT->RT)   ==   ROOTBT(B->RT-
>RT)){
                              B->LT->INFO = B->INFO;
                              B->INFO = '*';
                              B->RT->INFO = ROOTBT(B->LT->RT);
                              B->LT->RT->INFO = ROOTBT(B->RT->LT);
                              DESTROY(B->RT->LT);
                              DESTROY(B->RT->RT);
                         }
                         ELSE  IF(ROOTBT(B->LT->RT)  ==  ROOTBT(B-
>RT->LT)){
                              B->LT->INFO = B->INFO;
                              B->INFO = '*';
                              B->RT->INFO = ROOTBT(B->RT->LT);
                              B->LT->RT->INFO = ROOTBT(B->RT->RT);
                              DESTROY(B->RT->LT);
                              DESTROY(B->RT->RT);
                         }
                         ELSE  IF(ROOTBT(B->LT->LT)  ==  ROOTBT(B-
>RT->LT)){
                              B->RT->INFO = B->INFO;
                              B->INFO = '*';
                              B->LT->INFO = ROOTBT(B->LT->LT);
                              B->RT->LT->INFO = ROOTBT(B->LT->RT);
                              DESTROY(B->RT->LT);
                              DESTROY(B->RT->RT);
                         }
                         ELSE  IF(ROOTBT(B->LT->LT)  ==  ROOTBT(B-
>RT->RT)){
```

```
                    B->RT->INFO = B->INFO;
                    B->INFO = '*';
                    B->LT->INFO = ROOTBT(B->LT->LT);
                    B->RT->RT->INFO = ROOTBT(B->LT->RT);
                    DESTROY(B->RT->LT);
                    DESTROY(B->RT->RT);
                }
            }
        }
        REPLACEELEMBINTREE (LEFT(B));
        REPLACEELEMBINTREE (RIGHT(B));
    }
}

INT MAIN () {
    SETLOCALE(LC_CTYPE, "RUS");

    BINTREE BINARYTREE;
    STD::STRING BINTREESTRING;
    FOUT.OPEN("RESULT.TXT");

    INFILE >> BINTREESTRING;
    WHILE(BINTREESTRING != "END"){
        IF(BINTREESTRING.SIZE() > 1){
            FOUT << "ПРЕОБРАЗОВАНИЕ ДЕРЕВА ДЛЯ ФОРМУЛЫ: "
<< BINTREESTRING << STD::ENDL;
            BINARYTREE                                    =
ENTERBINTREEFORMULA(BINTREESTRING, 0);
            FOUT << "БИНАРНОЕ ДЕРЕВО (ПОВЕРНУТОЕ): " <<
STD::ENDL;
            DISPLAYBT (BINARYTREE, 1);
            FOUT << STD::ENDL;
            REPLACEELEMBINTREE(BINARYTREE);
            FOUT << "БИНАРНОЕ ДЕРЕВО (ПОВЕРНУТОЕ): " <<
STD::ENDL;
```

```
                    DISPLAYBT (BINARYTREE, 1);
                    FOUT << STD::ENDL << "ПОСТФИКСНАЯ ФОРМА: ";
                    PRINTLPK(BINARYTREE);
                    FOUT          <<          STD::ENDL          <<
"**********************************" << STD::ENDL;
                    INFILE >> BINTREESTRING;
                    DESTROY(BINARYTREE);
               }
               ELSE{
                    FOUT  <<  "ПРОСТЕЙШИЙ  СЛУЧАЙ  УСЛОВИЯ:  "  <<
BINTREESTRING << STD::ENDL;
                    INFILE >> BINTREESTRING;
               }
          }

          FOUT.CLOSE();
          INFILE.CLOSE();

          RETURN 0;
     }
```